Name: Anushka Harshavadan Nevgi
Experiment: 6. Implement a program to demonstrate the working of Top Down parsers.
Class: TY CSE A, 31

```
#include <iostream>
#include <string>
#include <deque>

using namespace std;
int n, n1, n2;

int getPosition(string arr[], string q, int size) {
   for (int i = 0; i < size; i++) {
      if (q == arr[i])
         return i;
   }
   return -1;
}

int main() {
   string prods[10], first[10], follow[10],
nonterms[10], terms[10];
   string pp_table[20][20] = {};

   cout << "Enter the number of productions: ";
   cin >> n;
   cin.ignore();

   cout << "Enter the productions" << endl;
   for (int i = 0; i < n; i++) {
      getline(cin, prods[i]);
      cout << "Enter first for " << prods[i].substr(3)
<< " : ";
      getline(cin, first[i]);
   }

   cout << "Enter the number of Terminals: ";
   cin >> n2;
   cin.ignore();

   cout << "Enter the Terminals" << endl;
   for (int i = 0; i < n2; i++) {
      cin >> terms[i];
   }
   terms[n2] = "$";
   n2++;

   cout << "Enter the number of Non-Terminals: ";
   cin >> n1;
   cin.ignore();

   for (int i = 0; i < n1; i++) {
      cout << "Enter Non-Terminal: ";
      getline(cin, nonterms[i]);
      cout << "Enter follow of " << nonterms[i] << "
: ";
      getline(cin, follow[i]);
   }

   cout << "\nGrammar" << endl;
   for (int i = 0; i < n; i++) {
      cout << prods[i] << endl;
   }

   for (int j = 0; j < n; j++) {
      int row = getPosition(nonterms,
prods[j].substr(0, 1), n1);
      if (prods[j].at(3) != '#') {
         for (int i = 0; i < first[j].length(); i++) {
            int col = getPosition(terms,
first[j].substr(i, 1), n2);
            pp_table[row][col] = prods[j];
         }
      } else {
         for (int i = 0; i < follow[row].length(); i++) {
            int col = getPosition(terms,
follow[row].substr(i, 1), n2);
            pp_table[row][col] = prods[j];
         }
      }
   }

   // Display Table
   for (int j = 0; j < n2; j++)
      cout << "\t" << terms[j];
   cout << endl;

   for (int i = 0; i < n1; i++) {
      cout << nonterms[i] << "\t";
      for (int j = 0; j < n2; j++) {
         cout << pp_table[i][j] << "\t";
      }
      cout << endl;
   }

   // Parsing String
   char c;
   do {
      string ip;
      deque<string> pp_stack;
      pp_stack.push_front("$");
      pp_stack.push_front(prods[0].substr(0, 1));

      cout << "Enter the string to be parsed: ";
      getline(cin, ip);
```

```
        ip.push_back('$');

        cout << "Stack\tInput\tAction" << endl;
        while (true) {
            for (int i = 0; i < pp_stack.size(); i++)
                cout << pp_stack[i];
            cout << "\t" << ip << "\t";

            int row1 = getPosition(nonterms,
pp_stack.front(), n1);
            int row2 = getPosition(terms,
pp_stack.front(), n2);
            int column = getPosition(terms, ip.substr(0,
1), n2);

            if (row1 != -1 && column != -1) {
                string p = pp_table[row1][column];
                if (p.empty()) {
                    cout << "\nString cannot be Parsed."
<< endl;
                    break;
                }
                pp_stack.pop_front();
                if (p[3] != '#') {
                    for (int x = p.size() - 1; x > 2; x--) {
                        pp_stack.push_front(p.substr(x, 1));
                    }
                }
                cout << p;
            } else {
                if (ip.substr(0, 1) == pp_stack.front()) {
                    if (pp_stack.front() == "$") {
                        cout << "\nString Parsed." << endl;
                        break;
                    }
                    cout << "Match " << ip[0];
                    pp_stack.pop_front();
                    ip = ip.substr(1);
                } else {
                    cout << "\nString cannot be Parsed."
<< endl;
                    break;
                }
            }
            cout << endl;
        }

        cout << "Continue?(Y/N) ";
        cin >> c;
        cin.ignore();
    } while (c == 'y' || c == 'Y');

    return 0;
}
```

```
Enter the number of productions: 5
Enter the productions
S->aXYb
Enter first for aXYb : a
X->c
Enter first for c : c
X->#
Enter first for # : #
Y->d
Enter first for d : d
Y->#
Enter first for # : #
Enter the number of Terminals: 4
Enter the Terminals
a b c d
Enter the number of Non-Terminals: 3
Enter Non-Terminal: S
Enter follow of S : $
Enter Non-Terminal: X
Enter follow of X : bd
Enter Non-Terminal: Y
Enter follow of Y : b

Grammar
S->aXYb
X->c
X->#
Y->d
Y->#
    a    b    c    d    $
S   S->aXYb
X        X->#    X->c    X->#
Y        Y->#        Y->d
Enter the string to be parsed: acdb
Stack    Input    Action
S$  acdb$    S->aXYb
aXYb$    acdb$    Match a
XYb$     cdb$     X->c
cYb$     cdb$     Match c
Yb$ db$ Y->d
db$ db$ Match d
b$  b$  Match b
$   $
String Parsed.
Continue?(Y/N) Y
Enter the string to be parsed: abd
Stack    Input    Action
S$  abd$     S->aXYb
aXYb$    abd$     Match a
XYb$     bd$ X->#
Yb$ bd$ Y->#
b$  bd$ Match b
$   d$
String cannot be Parsed.
```